
Clean Code

— Boston University CS 506 - Lance Galletti —

The Problem



The Problem

“Software Systems get replaced not when they wear out but when they crumble under their own weight because they have become too complex”



Why Does this Occur?

- english words + grammar != poetry
- Need analysis of the code itself - not just algorithms

Gems of Clean Code

- Structure
 - Before writing code ask “How will someone use this (or part of this) code?”. Minimize side effects
 - Do one thing
- Method
 - Top Down Approach
 - Bottom up Approach
 - Solve the problem first - then improve / refine
- General
 - Boring code is good code: keep it simple
 - Late Binding: start vague and refine (don't commit to specificity)
 - Many functions with small bodies > one function with large body
 - Check soundness by reading your code before testing

Clean Code

'Clean code' by Robert C. Martin

Github Gist Summary of the Book:

<https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29>

A Look into Functional Programming

“Functional Programs are mathematical expressions that are evaluated and reasoned about much like ordinary mathematical functions. As a result, these expressions are simple to analyze and compose for large-scale programs”

- Combinators on Lists
- An example coding exercise



Combinators on Lists

`xs = [1, 4, 9, 16, 25]`

- `map(xs, lam(x) => f(x)) = [f(1), f(4), f(9), f(16), f(25)]`

Ex:

`map(xs, lam(x) => x % 2) = [1, 0, 1, 0, 1]`

- Foldleft - idea of consuming the list via an accumulator

Ex:

`foldleft(xs, init, lam(x, acc) => f(x, acc))` `foldleft(xs, 0, lam(x, acc) => x + acc)`

- `acc = init`
- `acc = f(1, acc)`
- `acc = f(4, acc)`
- `acc = f(9, acc)`
- `acc = f(16, acc)`
- `acc = f(25, acc)`
- `Return acc`

1. `Acc = 0`
2. `Acc = 0 + 1 = 1`
3. `Acc = 1 + 4 = 5`
4. `Acc = 5 + 9 = 14`
5. `Acc = 14 + 16 = 30`
6. `Acc = 30 + 25 = 55`
7. `Return 55`

Combinators on Lists

$xs = [1, 4, 9, 16, 25]$ $ys = [1, 2, 3, 4, 5]$

- $\text{map2}(xs, ys, \text{lam}(x, y) \Rightarrow f(x, y)) = [f(1, 1), f(4, 2), f(9, 3), f(16, 4), f(25, 5)]$

Ex:

$\text{map2}(xs, ys, \text{lam}(x, y) \Rightarrow x * y) = [1, 8, 27, 64, 125]$

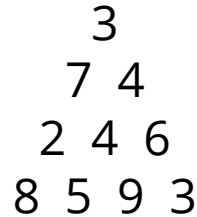
- Foldleft2 - consume both lists (like foldleft on zip)

Ex: $\text{foldleft2}(xs, ys, 1, \text{lam}(x, y, acc) \Rightarrow acc * (x / y))$

1. $acc = 1$
2. $acc = 1 * (1 / 1) = 1$
3. $acc = 1 * (4 / 2) = 2$
4. $acc = 2 * (9 / 3) = 6$
5. $acc = 6 * (16 / 4) = 24$
6. $acc = 24 * (25 / 5) = 120$
7. Return 120

Example: Max Path Sum

Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.

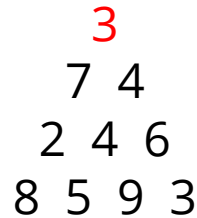


```
graph TD; 3 --> 7; 3 --> 4; 7 --> 2; 7 --> 4; 4 --> 4; 4 --> 6; 2 --> 8; 2 --> 5; 4 --> 9; 6 --> 3;
```

3
7 4
2 4 6
8 5 9 3

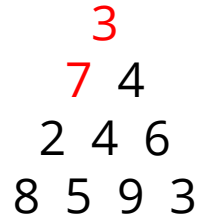
Example: Max Path Sum

Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.



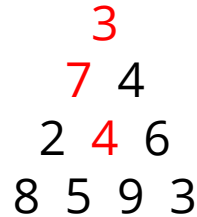
Example: Max Path Sum

Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.



Example: Max Path Sum

Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.



Example: Max Path Sum

Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.



Example: Max Path Sum

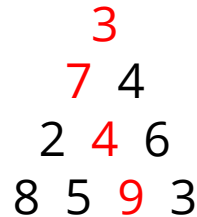
Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.



In the above example the max path sum is $3 + 7 + 4 + 9 = 23$

Example: Max Path Sum

Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.

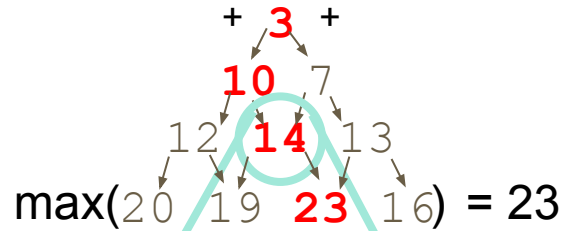


In the above example the max path sum is $3 + 7 + 4 + 9 = 23$

Brute Force: Find all paths and get the max.

Example: Max Path Sum

Our Algorithm



$$\max(10 + 4, 7 + 4)$$

Code Using Combinators

Triangle = list of lists. Can we use foldleft?

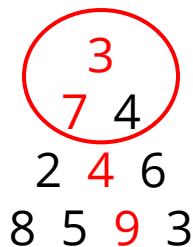
```
max( foldleft( triangle, [], lam(xs, acc) => myfold(xs, acc) ) )
```

3
7 4
2 4 6
8 5 9 3

Code Using Combinators

Triangle = list of lists. Can we use foldleft?

```
max( foldleft( triangle, [], lam(xs, acc) => myfold(xs, acc) ) )
```



Code Using Combinators

Triangle = list of lists. Can we use foldleft?

```
max( foldleft( triangle, [], lam(xs, acc) => myfold(xs, acc) ) )
```

3
7 4
2 4 6
8 5 9 3

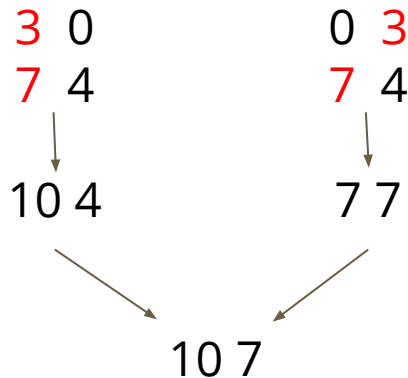
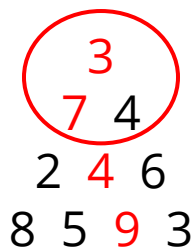
3 0
7 4
↓
10 4

0 3
7 4
↓
7 7

Code Using Combinators

Triangle = list of lists. Can we use foldleft?

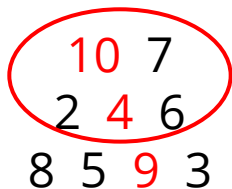
max(foldleft(triangle, [], lam(xs, acc) => myfold(xs, acc)))



Code Using Combinators

Triangle = list of lists. Can we use foldleft?

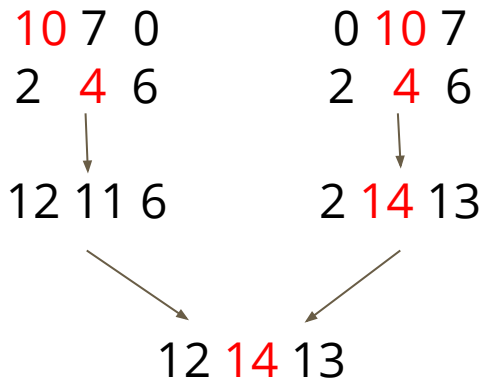
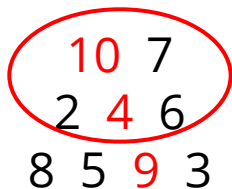
```
max( foldleft( triangle, [], lam(xs, acc) => myfold(xs, acc) ) )
```



Code Using Combinators

Triangle = list of lists. Can we use foldleft?

max(foldleft(triangle, [], lam(xs, acc) => myfold(xs, acc)))



Code Using Combinators

Triangle = list of lists. Can we use foldleft?

```
max( foldleft( triangle, [], lam(xs, acc) => myfold(xs, acc) ) )
```

```
myfold(xs, acc) :
```

```
option1 = map2( [0, acc...], xs, lam(x, y) => x + y )
```

```
option2 = map2( [acc..., 0], xs, lam(x, y) => x + y )
```

```
return map2( option1, option2, lam(x, y) => if x > y then x else y )
```


Python Examples

myfolder

```
|— main.py  
|— data.txt
```

main.py

```
nums = []
```

```
with open("../data.txt", "w+") as f:
```

```
    lines = f.readlines()
```

```
    for line in lines:
```

```
        nums += [int(x) for x in line.split(",")]
```

```
print(sum(nums))
```

data.txt

0,1,2,3,4

What is the output of this code?

- a. Prints "10"
- b. Raises an Error
- c. Prints "0"

Python Examples

```
class Bank:
    def __init__(self, balance):
        self.balance = balance

    def is_overdrawn(self):
        return self.balance < 0

myBank = Bank(100)
if myBank.is_overdrawn :
    print("OVERDRAWN")
else:
    print("ALL GOOD")
```

What is the output of this code?

- a. Prints "OVERDRAWN"
- b. Prints "ALL GOOD"
- c. Raises an Error

Python Examples

```
for i in range(4):  
    print(i)  
    i = 10
```

What is the output of this code?

- a. Prints "0"
- b. Prints "0 1 2 3"
- c. Raises an Error

```
some_string = "what"  
some_dict = {}  
for i, some_dict[i] in enumerate(some_string):  
    i = 10  
  
print(some_dict)
```

What is the output of this code?

- a. Prints "{}"
- b. Prints "{0: 'w', 1: 'h', 2: 'a', 3: 't'}"
- c. Raises an Error

<https://book.pythontips.com/en/latest/enumerate.html>

Python Examples

```
row = [""] * 3 # row i["", ""]
board = [row] * 3
print(board) # [["", "", ""], [ "", ""], [ "", ""]]
print(board[0]) # [ "", "" ]
print(board[0][0]) # ""
board[0][0] = "X"
print(board)
```

What is the output of this code?

- a. Prints "[['X', '', ''], ['', ''], ['', '']]'"
- b. Prints "[["", "", ""], ["", ""], ["", ""]]"
- c. Prints "[['X', '', ''], ['X', '', ''], ['X', '', '']]"

Python Examples

```
funcs = []
results = []
for x in range(3):
    def some_func():
        return x
    funcs.append(some_func)
    results.append(some_func()) # note the function call here

funcs_results = [func() for func in funcs]
print(results) # [0,1,2]
print(funcs_results)
```

What is the output of this code?

- a. Prints "[0,1,2]"
- b. Prints "[2,2,2]"
- c. Prints "[]"
- d. Raises an Error

Python Examples

You can find more such Python examples here [1]

[1] <https://github.com/satwikkansal/wtfpython>